# Affective Computing - Programming Assignment 2

## Objective

Your task is to extract a set of prosodic correlates (i.e. suprasegmental speech parameters) and cepstral features from speech recordings. Then, an emotion recognition system is constructed to recognize happy versus sad emotional speech (a quite easy two class problem) using a simple supervised classifier training and testing structure.

The original speech data is a set of simulated emotional speech (i.e. acted) from ten speakers speaking five different pre-segmented sentences of roughly 2-3 seconds in two different emotional states (happy and sad) totaling 100 samples. Basic prosodic features (i.e. distribution parameters derived from the prosodic correlates) are extracted using a simple voiced/unvoiced analysis of speech, pitch tracker, and energy analysis. Another set of Mel-Frequency Cepstral Coefficients (MFCC) features are also calculated for comparison.

To produce speaker independent and content dependent emotion recognition case (i.e. while a same persons samples are not included in both training and testing sets the same sentences are spoken in both the training and the testing sets) that could correspond to a public user interface with specific commands.

Support Vector Machine (SVM) classifiers are trained. A random subset of 1/2 of the available speech data (i.e. half of the persons) is used to train the emotion recognition system, first using a set of simple prosodic parameter features and then a classical set of MFCC derived features. The rest of the data (the other half of the persons) is then used to evaluate the performances of the trained recognition systems.

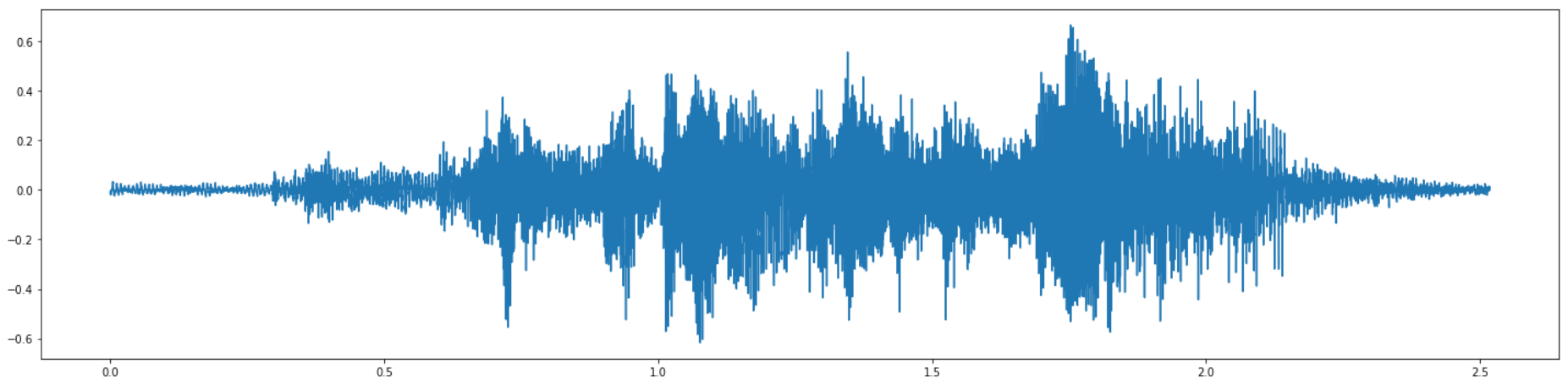
# Task 0. Preparation

Downsample the 'speech_sample' from the original Fs of 48 kHz to 11.025 kHz using `scipy.signal.resample()` (https://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.signal.resample.html) function.

**Steps**:

1. Load data 'speech_sample' from file *lab2_data.mat*. It is better to make sure the sample is a 1-D time series.
2. Declare the sampling frequency of the original signal, and your new sampling frequency.
3. Resample the signal using `scipy.signal.resample()` (https://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.signal.resample.html)
4. Visualize the resampled signal. Please make a appropriate time vector as the x axis.

In [11]:
```
# 1. Load the 'speech_sample'


# 2. Declare the source sampling frequency, and the target sampling frequency.
#    2.1 Source sampling frequency

#    2.2 Target sampling frequency
# Target frequency

# 3. Downsample the speech sample

# 4. Visualize the downsampled speech signal.
#    4.1 Creating the corresponding time vector, whose length is same to the length to the given signal.
#        You can use np.linspace() function to perform this. For example

#    4.2 Plot your result

```




# Task 1. Feature Extraction

## Task 1.1 MFCC calculations using the provided sample speech signal.

**Steps**:

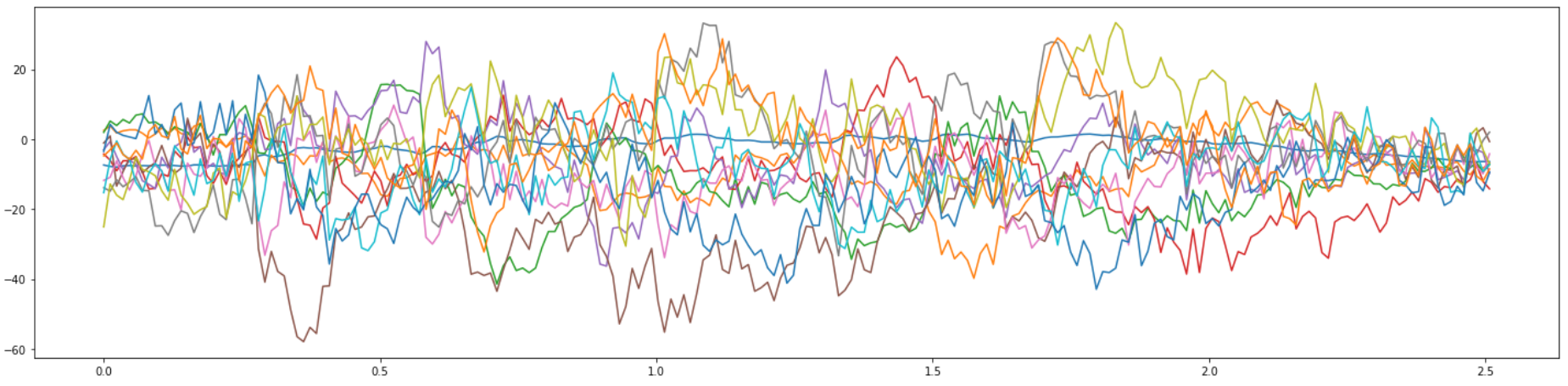
1. Pre-emphasize the resampled signal by applying a high pass filter, using the `scipy.signal.lfilter()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html) function. Apply a pre-emphasis filter
$$H(z) = 1 - \alpha \times z^{-1}$$
with α=0.98 to emphasize higher frequencies in your downsampled speech signal (Tip: use `scipy.signal.lfilter` (https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.signal.lfilter.html)).

   Hint for defining the filter: you will provide two vectors **b** and **a** to define the filter, **a** for the denominator and **b** for the numerator. So finally your filter will be defined as

$$H(z) = \frac{b[0]z^0 + b[1]z^{-1} + \ldots + b[i]z^{-i} + \ldots}{a[0]z^0 + a[1]z^{-1} + \ldots + a[i]z^{-i} + \ldots}$$

2. Extract the 12 mfcc coefficient by using the `python_speech_features.mfcc()` (http://python-speech-features.readthedocs.io/en/latest/) function.
    A. **The `python_speech_features.mfcc()` (http://python-speech-features.readthedocs.io/en/latest/) function has its internal pre-emphasis functionality. However, we do the pre-emphasis beforehand in order to have a better understanding on it.**
3. Visualize the 12 mfcc coefficient contours. Please also make the corresponding time vector as the x axis.
4. Calculate the mean of each contour using `numpy.mean(axis=axis)` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html).

In [2]:
```
from scipy.signal import lfilter
from python_speech_features import mfcc
# 1. Pre-emphasize your resampled signal.
#    1.1 Define the polynomial of your fitler
#        filter coefficients b, which is the numerator
#        filter coefficients a, which is the denominator


#    1.2 Apply the filter on the signal

# 2. Extract the mfcc coefficient by callying the mfcc() function
     # remeber to set the pre-emphasize argument to 0 since the signal has been pre-emphasized.


# 3. Plot the 12 mfcc contours
#    3.1 Create the time vector for the MFCC contour, again, using the np.linspace() function.
#        However, please note the length of the resulted mfcc contour is different to the sample length.
#        Thus, the length of your time vector here should be the length of the mfcc contour
#        The ending point can be computed by the: a) number of mfcc samples, and b) time length of samples used
#        for calculating one mfcc sample.
#
#        For example, here we set the frame increment length is 128 samples, corresponding roughly to 11.6ms frames
#        at Fs=11025Hz. So for inputs of the np.linspace() function, the ending point is mfccNum * 11.6ms around.


# 4. Calculate the mean for each contour.

```




```
[ -2.16803659  -7.55293839 -11.45023659  -9.15052974  -4.9467118
 -18.90732587  -9.39367434  -0.93334243   1.59932744  -7.89636553
 -13.94829969  -0.62393599]
```

**Question 1. Why do we need to pre-emphasize the speech signal before computing the MFCC feature?**

**Your answer:**

In [ ]:
```
1
```

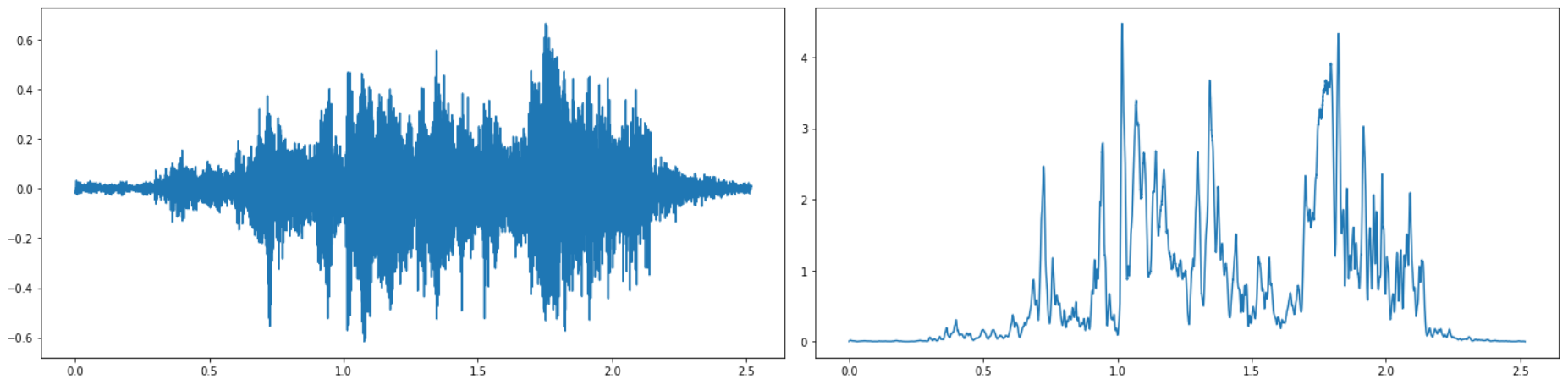
## Task 1.2 Extract the Intensity/Energy parameter

Firstly, please calculate the short time energy (STE) of the downsampled 'speech_sample' using the squared signal $x(t)^2$ and a 0.01s hamming window frames (Note! the extra length of the window. Clip half a window length from the beginning and at the end). Then calculate 5 distribution parameter features from the utterance (the signal).

**Steps**:

1. Define a hamming window using the `scipy.signal.hamming()` (https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.signal.hamming.html) function. The window length is the number of frames in 0.01s.
2. Apply the hamming window to convolve the squared signal, using the `scipy.signal.convolve()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html) function. The convolution result is the short time energy (STE) controu.
3. Clip half window of frames from the begining and ending of STE contour.
4. Visualize the resulted STE controur. Please also include the time axis
5. Calculating the following 5 distribution parameter feature from the STE contour:
    A. Mean, using the `numpy.mean(https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.mean.html)` () function.
    B. Standard Deviation (SD), using the `numpy.std()` (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.std.html) function.
    C. 10% percentile, using the `numpy.percentile()` (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.percentile.html) function.
    D. 90% percentile, using the `numpy.percentile()` (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.percentile.html) function.
    E. Kurtosis, using the `scipy.stats.kurtosis()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kurtosis.html) function.

In [3]:
```
# 1. Define a hamming window
#    1.1 Calculate the window length, which the number of frames within 0.01s

#    1.2 Define the hamming window using signal.hamming()


# 2. Calculate the short time energy (STE) contour by convolve the hamming window and the squared signal,
#    using the scipy.signal.convolve() function


# 3. Clip half window of frames from both the beginning and end of the STE contour


# 4. Visualize the final STE contour.
#    4.1 Create the time vector for x-axis

#    4.2 Visualize the STE contour


# 5. Calculate the 5 distribution parameter feature the of STE contour


```




```
[0.7478381956040606, 0.9053013037469263, 0.007376874331307642, 2.0824014793268333, 1.9364141357722326]
```

### Question 2. Why do we need to clip out half frame from both begining and end of the STE contour (you can find the answer by understanding how convolution is implemented)?

### Your Answer:

In [ ]:
```
1
```

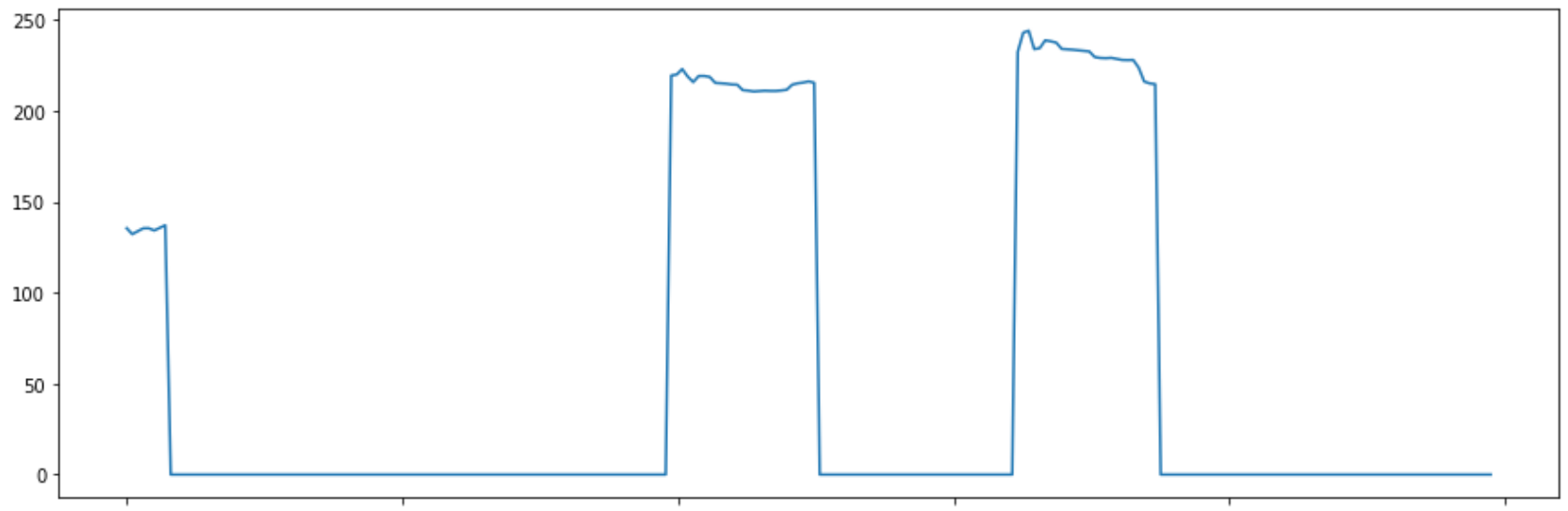
### Task 1.3. Extract the Pitch/F0 feature

**Steps**:

1. Extract the Pitch/F0 contour of the resampled speech signal using the **getF0()** function in 0.01s frames. The function is provided in the *f0Lib.py* file.
2. Visualize the F0 contour (including the time axis).
3. Extract the 5 distribution parameter features of the extracted F0 countour.

In [4]:
```
# 1. Extract the F0 contour

# 2. Visualize the F0 contour
#    2.1 The time vector can be acquired from the the third returned value of the getF0() function
#        For example T_ind. The time vector can be computed by dividing the the first column of T_ind
#        with the sampling frequency

#    2.2 Visulize the F0 contour.


# 3. Calculate these distribution parameter features

```

```
[51.76557834234848, 92.16529162886495, 0.0, 219.59546850800427, -0.32757516607828796]
```




## Task 1.4. Extract the Rhythm/Durations parameter

**Steps**:

1. Perform a Voiced/Unvoiced speech segmentation of speech signal. Tip: Unvoiced frames are marked with 0 F0 values, you can find the voiced frames (i.e. F0 > 0) using `numpy.where()` (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.where.html).
2. From the segmentation, calculate the means and SDs of both Voiced and Unvoiced segment lengths (i.e. voiced segment mean length, SD of voiced segment lengths, unvoiced segment mean length, SD of unvoiced segment lengths).
3. Calculate also the voicing ratio, i.e. the ratio of voiced segments versus total segments (Tip: You can do this simply by using the frames).

In [5]:
```
# 1. Segmenting the voiced and unvoiced speech segements.
#    1.1 Example on extracting voiced segment lengths


##########################################################################
##########################################################################
#    1.2 Extract unvoiced segment lengths.


# 2. Calculate the means and SDs of both Voiced and Unvoiced segment lengths


# 3. Calculate the voicing ratio.


```

```
[20.333333333333332, 62.666666666666664, 8.73053390247253, 22.48456260538674, 0.24497991967871485
```

## Task 1.5. Check your extracted feature

1. Print your calculated 12 MFCC coefficients.
2. Print the distribution parameter feature of the STE contour.
3. Print the distribution parameter feature of the F0 contour.
4. Print the 5 prodosic features: **mean** and **std** of lengths of **voiced speeches** and **unvoiced speech**, as well as the **voicing ratio**.

In [6]:
```
# 1. Print the 12 MFCC coefficients
print (mfccContour.mean(0))
# 2. Print the distribution paremeter feature of the STE contour
print (steFeature)
# 3. Print the distribution parameter feature of the F0 contour
print (f0Feature)
# 3. Print the 5 prodosic features
print (rythumFeature)
```

```
[ -2.16803659  -7.55293839 -11.45023659  -9.15052974  -4.9467118
 -18.90732587  -9.39367434  -0.93334243   1.59932744  -7.89636553
 -13.94829969  -0.62393599]
[0.7478381956040606, 0.9053013037469263, 0.007376874331307642, 2.0824014793268333, 1.9364141357722326]
[51.76557834234848, 92.16529162886495, 0.0, 219.59546850800427, -0.32757516607828796]
[20.333333333333332, 62.666666666666664, 8.73053390247253, 22.48456260538674, 0.5]
```

# Task 2. Speech Emotion Classification

In this part ,you we will the `sklearn.svm` (http://scikit-learn.org/stable/modules/svm.html) library to perform the speech signal classification. The **'training_data_proso'** and **'training_data_mfcc'** matrixes contain the calculated prosodic features for the training set (9 features in each row representing a speech sample) and MFCC derived features (12 features) respectively. The **'training_class'** group vector contains the class of samples: 1 = happy, 2 = sad; corresponding to the rows of the training data matrices.

In thie part, you will get familiar to three kinds of classifiers, namely the SVM classifier, the Random Forest classifier, and the the Neural Network classifer (a multi-layer perceptron).

## Task 2.1. Train and evaluate your SVM classifiers

**Steps**:

1. Load training data.
2. Train an SVM with the prosody data using the **'training_data_proso'** features and a **3rd order polynomial** kernel.
3. Train an SVM with the MFCC data using the **'training_data_mfcc'** features and a **3rd order polynomial** kernel.
4. Load testing data
5. Calculate the average classification accuracy for the training data (**'training_data_proso'** and **'training_data_mfcc'**) using the corresponding prosody and MFCC trained SVMs.
6. Calculate the average classification accuracy for the testing data (**'testing_data_proso'** and **'testing_data_mfcc'**) using the corresponding prosody and MFCC trained SVMs.
7. Print the four accuracies you have calculated.
8. Plot confusion matrices for the training and testing data for both classifiers.

In [7]:
```
# 4. Print the four accuracies.
print acc_p_tr   # accuracy on the prosodic training data
print acc_m_tr   # accuracy on the mfcc training data
print acc_p_te   # accuracy on the prosodic testing data
print acc_m_te   # accuracy on the mfcc testing data

from sklearn.metrics import confusion_matrix
# confusiuon matrix on the prosodic training data
# confusiuon matrix on the mfcc training data
# confusiuon matrix on the prosodic testing data
# confusiuon matrix on the mfcc testing data
```

```
1.0
0.96
0.84
0.86
[[25  0]
 [ 0 25]]
[[23  2]
 [ 0 25]]
[[21  4]
 [ 4 21]]
[[22  3]
 [ 4 21]]
```

In [ ]:
```

```

In [ ]:
```

```

## Task 2.2 Train and evaluate a Random Forest Classifier and evaluate it

Please train one Random Forest classifer for each of the Prosodic and MFCC feature using the `sklearn.ensemble.RandomForestClassifier()` (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html), and print the classification accuracies and confusion matrices.

In [8]:
```
# 4. Print the four accuracies.
print acc_p_tr   # accuracy on the prosodic training data
print acc_m_tr   # accuracy on the mfcc training data
print acc_p_te   # accuracy on the prosodic testing data
print acc_m_te   # accuracy on the mfcc testing data

from sklearn.metrics import confusion_matrix
# confusiuon matrix on the prosodic training data
# confusiuon matrix on the mfcc training data
# confusiuon matrix on the prosodic testing data
# confusiuon matrix on the mfcc testing data
```

```
1.0
0.98
0.84
0.82
[[25  0]
 [ 0 25]]
[[24  1]
 [ 0 25]]
[[21  4]
 [ 4 21]]
[[21  4]
 [ 5 20]]
```

### Task 2.3 Train and evaluate a Neural Network Classifier and evaluate it

Please train a multi-layer perceptron for each of the Prosodic and MFCC feature using the `sklearn.neural_network.MLPClassifier()` (http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html), and print the classification accuracies and confusion matrices.

In [6]:
```
# 4. Print the four accuracies.
print acc_p_tr   # accuracy on the prosodic training data
print acc_m_tr   # accuracy on the mfcc training data
print acc_p_te   # accuracy on the prosodic testing data
print acc_m_te   # accuracy on the mfcc testing data

from sklearn.metrics import confusion_matrix
# confusiuon matrix on the prosodic training data
# confusiuon matrix on the mfcc training data
# confusiuon matrix on the prosodic testing data
# confusiuon matrix on the mfcc testing data
```

```
1.0
1.0
0.82
0.86
[[25  0]
 [ 0 25]]
[[25  0]
 [ 0 25]]
[[20  5]
 [ 4 21]]
[[25  0]
 [ 7 18]]
```

## Task 3. Speech Emotion Classification using SVM, Subject Independent

Generate a person independent 10-fold cross-validation (CV) estimate of the emotion recognition system performance.

- Join the training/testing data matrices and the class vectors. Combine also the ‘training_data_personID’ and ‘testing_data_personID’ vectors that are needed to make the CV folds.
- Construct the CV folds by training ten SVMs. For each SVM nine persons’ data is used as the training set (i.e. 90 samples) and one persons’ samples are kept as the test set (i.e. 10 samples) for the respective fold (i.e. each SVM has different persons’ samples excluded from the training set). Test each ten trained SVMs by using the corresponding one held-out persons’ samples and then calculate the average classification performances for each fold.
- Calculate the mean and SD of the ten CV fold performances to produce the final CV performance estimate of the emotion recognition system.

In [10]:
```


#accs_prosofic.mean(), accs_prosofic.std(), accs_mfcc.mean(), accs_mfcc.std()])
```

```
[0.8600000000000001, 0.11135528725660043, 0.8800000000000001, 0.15362291495737218]
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```